Do Rapid Releases Affect Bug Reopening? A Case Study of Firefox

Rodrigo Souza
Office of Information Technology
Federal University of Bahia, Brazil
Email: rodrigo@dcc.ufba.br

Christina Chavez
Computer Science Department
Federal University of Bahia, Brazil
Email: flach@dcc.ufba.br

Roberto A. Bittencourt
State University of Feira de Santana, Brazil
Email: roberto@uefs.br

***Abstract*—Large software organizations have been adopting rapid release cycles to deliver features and bug fixes earlier to their users. Because this approach reduces time for testing, it raises concerns about the effectiveness of quality assurance in this setting. In this paper, we study how the adoption of rapid release cycles impacts bug reopening rate, an indicator for the quality of the bug fixing process. To this end, we analyze thousands of bug reports from Mozilla Firefox, both before and after their adoption of rapid releases. Results suggest that the bug reopening rate of versions developed in rapid cycles was about 7% higher. Also, as a warning to the software analytics community, we report contradictory results from three attempts to answer our research question, performed with varying degrees of knowledge about the Firefox release process.**

I. Introduction

The internet has made easier to distribute software, effectively increasing the competition between software organizations. The so-called "internet time" [1] pushed organizations to release new features at a faster pace. As a result, in the last decade, projects such as Firefox and Unity3D started moving from a 12–18 month release cycle to a shorter, 1–3 month release cycle.

Although the main motivation for moving to rapid (or short) release cycles is time-to-market, some people argue that it also helps improve software quality. As developers are expected to always implement potentially shipable source code, testing efforts can be more frequent, so hopefully less bugs reach end users. On the other hand, short cycles may lead to tighter deadlines for testing, which may ultimately lead to less stable versions being released to the public.

Whether rapid releases actually reduce bugs is subject to discussion. Khohm et al. [2], for instance, found no significant difference in the number of bugs created each day when comparing traditional and short releases at Mozilla (although the median number of bugs increased a little under short releases).

In this paper we explore a related, but different question: how do rapid release cycles affect bug reopening? Bug reopening is what happens when, after a bug report is resolved, someone discovers that the resolution was inappropriate or incomplete.

Bug reopening is undesirable because it leads to rework. While it cannot be completely avoided, a high bug reopening rate may indicate an unstable development process [3].

Despite its importance, there is still little empirical evidence on how bug reopening is influenced by the software development process. In this paper, we investigate three research questions on the relationship between rapid releases and bug reopening:

RQ1: Do rapid releases impact the bug reopening rate? We expect that, with short release cycles, bug reopening would be more frequent. The rationale is that, in that case, bugs would be fixed faster, and might not cover corner cases. Also, there would be less time for testing.

RQ2: Do rapid releases impact the number of bugs reopened due to failing automated tests? If a bug is reopened due to a failing automated test, it means that the initial manual testing and core review were not successful. We expect that, because rapid releases leave less time for comprehensive manual testing, bug fixes would fail automated tests more often.

RQ3: Do rapid releases impact the time it takes for a bug to be reopened? The time between a bug fix and its reopening, also known as *latent time* [4], measures how fast faulty fixes are detected. Smaller latent times mean that end users are less likely to get software with bugs that were believed to be fixed. We expect that, when software is released more often, faulty bug fixes would be discovered faster by plugin developers and early users, leading to a shorter latent time.

To conduct this study, we performed statistical analyses on bug reports and commit logs from Mozilla Firefox, an open source web browser used by half a billion people worldwide[1]. Mozilla recently moved from a traditional, planned release cycle, to a rapid, 6-week release cycle.

The results suggest that (1) the bug reopening rate increased 7% under rapid releases; (2) the number of bugs reopened due to failing automated tests did not change significantly; and (3) long-lived faulty bug fixes tended to be discovered earlier under rapid releases. Although it is not possible to generalize from a single case, such results can provide insight for further studies.

The word "bug" is used in a broad sense, meaning any problem that was documented in a bug report. A significant proportion of bug reports do not refer to corrective maintenance tasks [5]; they may refer instead to performance im-

[1] https://blog.mozilla.org/press/ataglance/

provements, adaptive maintenance, refactoring, documentation, among other tasks [6].

In this work, we also found that pursuing software engineering research using a repository mining approach may lead to systematic bias. Because of incomplete information about the release process and how bug reopening is reported, the analyses had to be repeated three times. Each analysis used additional data and led to a different conclusion. We describe how we were able to identify flaws in the first two analyses after gathering developer feedback and information from exploratory data analysis.

II. Related Work

Both bug reopening and the effects of rapid releases on software quality are underexplored topics in the scientific literature. To the best of our knowledge, there is no empirical study that assesses the effects of rapid releases on bug reopening[2]. For this reason, in this section we present previous work about rapid releases and bug reopening.

A. Short Release Cycles

Baskerville and Pries-Heje [1] interviewed employees from companies that adopted short release cycles in 2000. According to an interviewee, short release cycles limit reuse and systematic thinking. Also, the authors found that, in the companies they studied, process quality and product quality were sacrificed in the name of meeting customers' vague requirements.

Khomh et al. [2] studied the effect of short release cycles on the quality of Mozilla Firefox, assessed by three metrics: number of post-release bugs created per day, crash rate, and uptime (i.e., the time between a user starting up Firefox and experiencing a failure). Comparing data from traditional and rapid releases, they found no significant difference in the number of post-release bugs and crash rate. The uptime, on the other hand, was significantly lower in releases developed in short cycles, meaning that the software crashed earlier during its usage.

Mäntylä et al. [7] studied the effects of rapid releases on test case executions at Mozilla. They found that, under rapid releases, testing was more focused: the number of test executions increased, but the scope was reduced. Instead of running the whole test suite, engineers narrowed the scope to high risk features and regressions. The authors also noted that, to keep up with testing needs, more specialized testers were hired.

Public opinions on websites show that Mozilla's transition to rapid releases aroused mixed reactions. Some users complained that frequent updates were annoying; others experienced intermittent compatibility problems [8]; early versions had noticeable performance issues [9]. Some users report that such problems were mitigated in recent Firefox versions.

[2]Although no systematic review was carried out, a thorough search was conducted just before writing the paper. Also, the authors have been looking extensively for empirical papers about bug reopening (there are few papers on the topic) and, through an online service, are notified whenever a new paper cites one of the papers that were found to be about bug reopening.

B. Bug Reopening

Shihab et al. [10], [11] developed a decision tree model, based on features about the bug report, the bug fix, and human factors, to predict which bugs would be reopened. They found that among the top predictors of bug reopening are the component in which the bug was found and the time needed to to submit the first fix to the bug.

In a partial replication of Shihab's work, Zimmermann et al. [3] found that bugs reported by users are more likely to be reopened than those discovered through code review or static analysis, supposedly because those reported by users are harder to reproduce and tend to be more complex. Other factors that favor reopening, according to the authors, include bug severity and geographical distribution of developers participating in the bug.

The authors also asked Microsoft engineers about the common causes for bug reopening. Responses included the difficulty to reproduce a bug, the misunderstanding of root causes, the lack of information in the initial report, the increase of the bug priority, incomplete fixes, and code integration problems.

Almossawi [12] analyzed 32 open source systems in the GNOME project and, using a logistic regression model, concluded that bugs located in code with high cyclomatic complexity are more likely to be reopened. Jongyindee et al. [13] found that bugs fixed by more experienced developers are less likely to be reopened.

Regarding the impact of reopened bugs, the literature reports reopening rates as low as 1.4% [12] and as high as 11.7% [13], obtained from open source projects. Also, reopened bugs are said to have a life cycle from 2 to 5 times higher [10], [13] than regular bugs, and to involve the participation of 40% more developers [13].

To the best of our knowledge, no empirical study was ever conducted to assess the impact of a process change in the bug reopening rate. In this paper, we investigate the impact of a specific change, the adoption of rapid releases, in bug reopening.

III. Release Engineering at Firefox

The objective of this section is to explain the development process adopted by Mozilla, as well as to describe how specific parts of the process are recorded using software development tools. The description presented in this section is based on semi-structured interviews with a Mozilla engineer, and on information found in Mozilla's wiki [14] and in Mozilla's draft on process documentation for software release mechanics [15]

Before March 2011, Firefox had been developed according to a traditional release schedule: features for the upcoming version were developed along with bug fixes and minor updates for the current stable release. Major features would only be delivered to users with the release of a major version, which ocurred when the planned features were implemented and tested. In practice, a new major version used to take 12 to 18 months to be released [14].

Partly because of Google's Chrome competition [8], a web browser that is frequently updated with new features, Mozilla

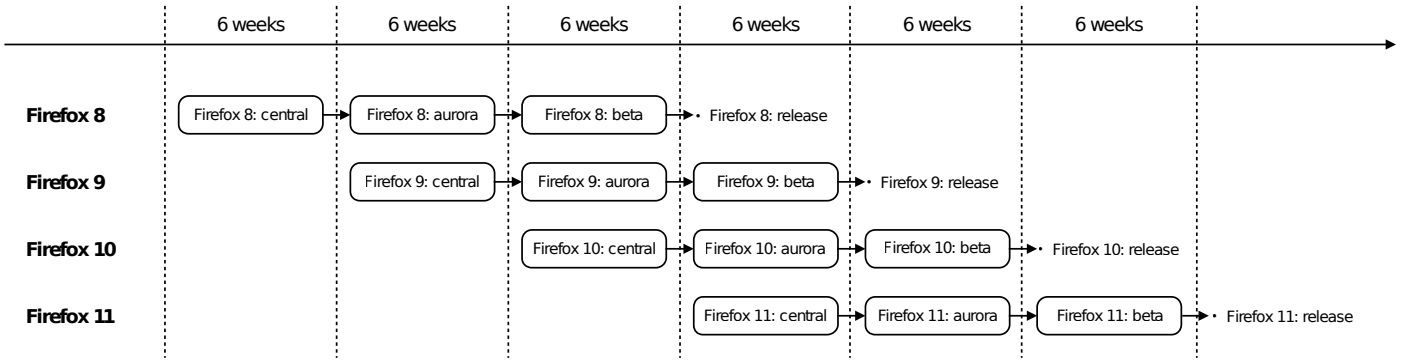

Fig. 1. Mozilla's train model of software release

decided to move to a rapid release cycle in which new versions would be released every 6 weeks.

The release process adopted by Mozilla is called "train model". In this process, each new feature or bug fix flows through three source code repositories – central (or nightly), aurora (or experimental), and beta – before being integrated to the source code of the next official release [15]. Each repository is more stable than the former, and latter repositories have stricter change policies. Also, each repository reaches a wider audience; e.g., while nightly builds are expected to be run by 100,000 developers and very early adopters, the number of users that run final releases is in the order of hundreds of millions.

Every 6 weeks occurs the merge day, when patches in central are merged into aurora, patches in aurora are merged into beta, and patches in beta become part of the next release. Hence, a new feature can be delivered to users in 12 to 18 weeks, as patches flow through nightly, aurora and beta channels.

In order to release a new version every 6 weeks, multiple versions are developed in parallel. While a version is being stabilized in the aurora repository, the central repository receives features and bug fixes for the subsequent version.

Figure 1 shows a simplified schedule for the release of versions 8, 9, 10, and 11. It can be seen, in the fourth 6-week cycle, that when Firefox 8 is released, Firefox 11 starts to be developed in central, while Firefox 9 and 10 are being stabilized in aurora and beta to be released afterwards.

A. Detailed Process

At Mozilla, two tools are central to coordinate the development of features and bug fixes: the bug tracking system, Bugzilla[3], and the distributed version control system, Mercurial[4].

In Bugzilla, people can report bugs and then update bug reports with information regarding the bug and the bug fixing process, either by uploading files (screenshots, trace logs, patches) or by commenting and updating bug report fields. Each bug report has a status field, which can take values such as NEW, RESOLVED, VERIFIED, and REOPENED. For

[3]http://www.bugzilla.org/

[4]http://mercurial.selenic.com/

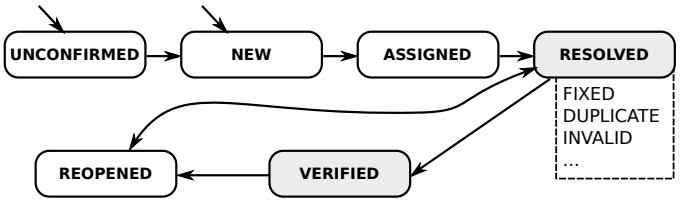

Fig. 2. Bug status and their transitions on Bugzilla

RESOLVED bugs, a resolution must be chosen among FIXED, INVALID, DUPLICATED and others.

Figure 2 shows typical bug status and transitions between them. A bug starts with status UNCONFIRMED, if reported by a regular user, or NEW, if reported by a trusted user. After that, the bug may be ASSIGNED to a developer, and then RESOLVED (with resolution FIXED, if the bug was solved). After successful testing, the status is changed to VERIFIED. Any bug marked RESOLVED or VERIFIED may be REOPENED if the initial resolution was inappropriate so it can be RESOLVED correctly.

Source code is kept in Mercurial repositories. There are separate repositories for central, aurora, beta, and release. Since late 2011, there is also a repository called inbound, where developers can commit code that is run against automated tests before being merged into central.

Figure 3 shows how people with distinct roles interact with each other through Bugzilla and Mercurial to fix bugs and test the bug fixes. There are at least four roles involved in the handling of bugs: the developer who writes a patch for the bug, the developer who reviews the patch, the build sheriff, who merges the patches into central depending on the outcome of automated tests, and the tester, who performs manual testing by running versions built nightly from central.

When a developer fixes a bug, he attaches the patch to the corresponding bug report in Bugzilla and asks a specific colleague to review it. If the reviewer approves the patch, the developer commits it to the inbound repository. Otherwise, the developer should improve his patch and start the process again.

Patches in inbound undergo automated testing, which includes unit, integration, and interface tests. If all tests pass, the build sheriff merges the patch into the central channel and

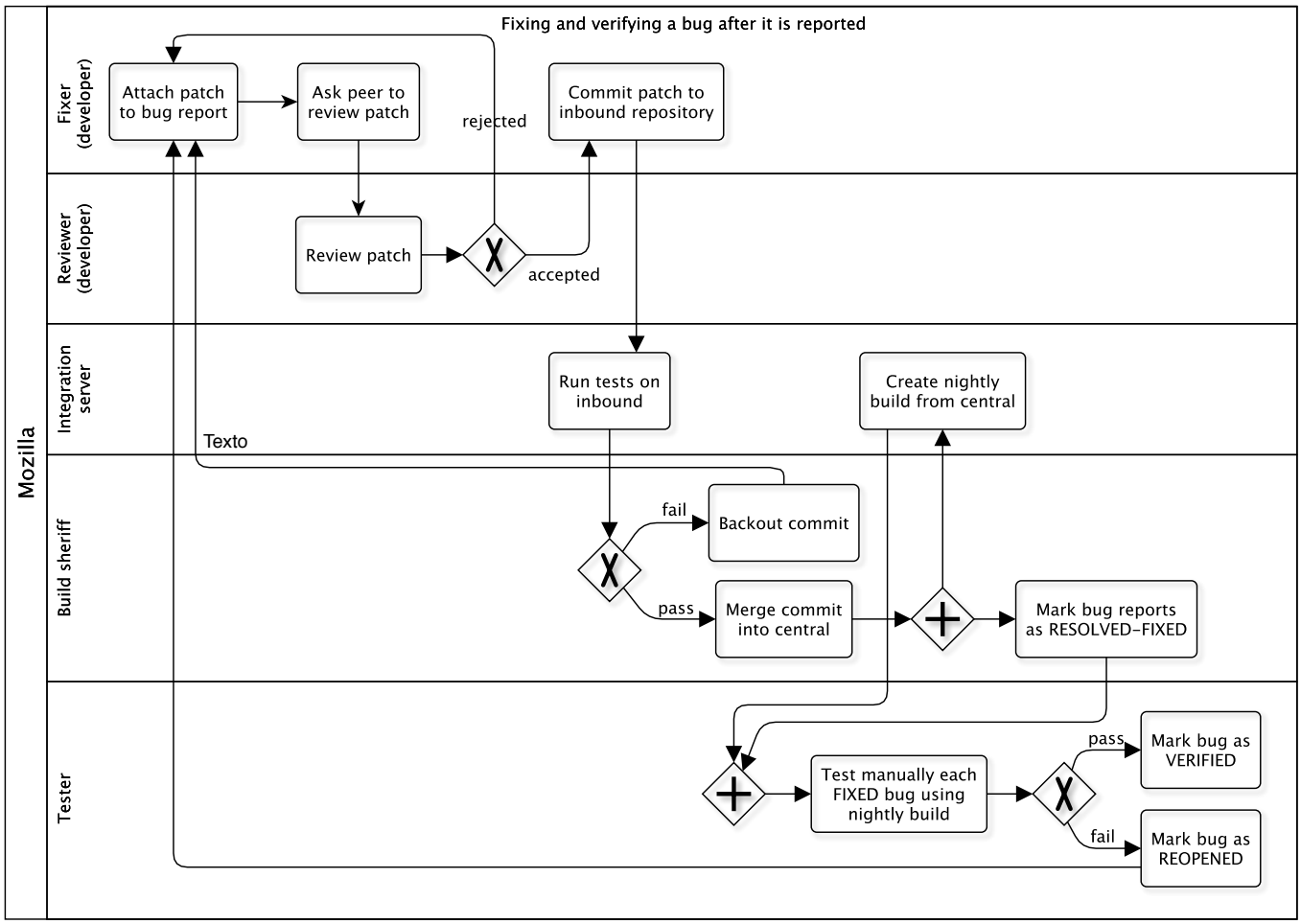

Fig. 3. Mozilla bug fixing process

changes the bug status to RESOLVED and the resolution to FIXED. Otherwise, the sheriff is responsible for discovering which commit broke the tests and for "backing out" the commit (i.e., creating another commit that reverts the problematic commit).

After that, a tester downloads a nightly build and manually tests it to check if all bugs marked as RESOLVED were actually resolved. If this is the case, the bug report is marked as VERIFIED; otherwise, it is marked as REOPENED.

When a version reaches aurora or beta, no new features are admitted. More comprehensive testing is performed, and engineers work to stabilize or even disable problematic features, while other contributors translate the user interface to several languages.

Although this description refers to the development process adopted by Mozilla after moving to rapid release cycles, it is similar to the process adopted before. The main difference is that, before rapid releases, the aurora, beta and inbound repositories did not exist — developers used to commit code directly to central, where the code was stabilized for the next release.

IV. Data Analysis

To assess whether the adoption of short release cycles led to an increase in the bug reopening rate, we analyze bugs reported for Firefox from 2009 to 2013, available in Mozilla's Bugzilla bug database. The data set was provided as a SQL database dump by a Mozilla engineer[5].

Figure 4 shows, in thick lines, the period being analyzed. The bugs are split, according to their creation date, into two periods: from 2009-06-30 to 2011-03-22, when versions 3.6 and 4.0 were developed using a traditional release lifecycle, and from 2011-09-27 to 2013-09-17, when versions 8 up to 27, which followed rapid releases, were developed. Version 4.0 was the last one developed with traditional releases; version 8 was the first that strictly followed the rapid release schedule (central, aurora, beta, release).

Although the Bugzilla database dump includes bugs created up to 2013-11-30, only bugs created before 2013-09-18 were considered. This gives at least 2 months of historical data for each bug, which is usually sufficient time for a bug to be reopened, if it needs to be [16].

[5]https://bugzilla.mozilla.org/page.cgi?id=researchers.html

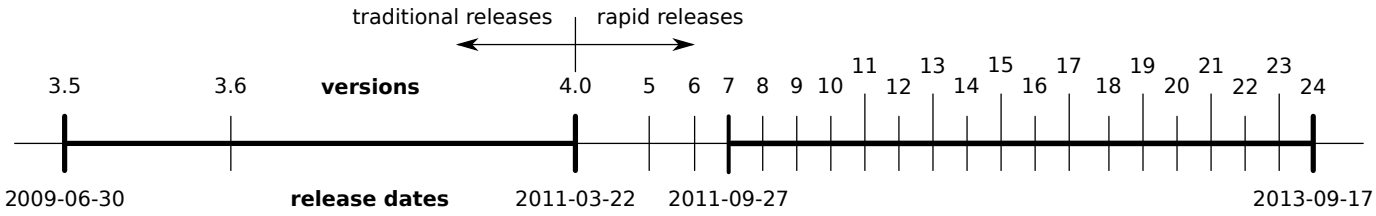

Fig. 4. Firefox release history and periods being studied

Only bugs that ended up with status RESOLVED and resolution FIXED were considered. The purpose is to avoid taking into account duplicate bug reports and issues that are not bugs.

To evaluate whether short releases impact bug reopening (RQ1), we determined, for each bug, whether it has been reopened when its resolution was FIXED. Bugs were not considered reopened if they had their status changed to REOPENED after they had been resolved as DUPLICATE, INVALID, etc.

Bugs were summarized in a contingency table, that counts the number of reopened and non-reopened bugs under each release type (traditional and rapid). Then, Fisher's exact test [17] was applied to the table to determine if there is an association between the two variables (reopening and release type).

A. First Analysis

At first, we tried to answer RQ1 by looking only at the REOPENED bug status (the commit log was not considered). Fisher's exact test yielded $p < 0.05$, meaning that an association between release type and bug reopening was found. Contrary to initial belief, though, the data showed that bugs created under rapid releases were about 50% *less likely* to be reopened.

After these preliminary results were sent to the Firefox development mailing list[6], a developer clarified that some bug reopenings are not recorded using the REOPENED status:

> Now we land stuff on mozilla-inbound, but still don't mark bugs as RESOLVED FIXED until mozilla-inbound gets merged to mozilla-central. And we only merge changesets that pass all the tests, so stuff now "bounces" without ever resolving (and hence reopening) the bug.

Another developer pointed out that these cases could be detected by looking for "backout" commits, i.e., commits that revert a previous change.

B. Second Analysis

By reading a small sample of bug reports, we noticed that some comments contained the expression "backout" or some variation (such as "back out", "backed out", and "backing out"). The analysis was repeated, this time treating all references to "backout" in bug reports as reopening.

[6]https://mail.mozilla.org/listinfo/firefox-dev

The difference between reopening rates, in this case, was not statistically significant. By reading a larger sample of comments, though, we noticed that many comments that contain "back out" are referencing other bugs, or just discussing the possibility of backing out the bug.

Also, a developer mentioned that the fact the a bug is marked as FIXED does not always mean that it was fixed by means of a source code patch. Sometimes bugs are fixed by resolving another bug, or by some procedure that does not involve changing the source code. For example, if there is a problem with the testing infrastructure, some automated tests may fail, resulting in a bug being reported. After fixing the infrastructure (e.g., by restarting the server or updating a library), the bug can be considered FIXED, although no changes needed to be made to the source code.

C. Final Analysis

The two previous attempts had flaws that were uncovered by either exploring the data or by getting feedback from developers:

- not all bug reopenings are recorded using Bugzilla's REOPENED status;
- bug reopening is sometimes materialized in a "backout" commit;
- not all bugs resolved as FIXED involved changing the source code.

The data analysis process was improved to overcome those flaws. Figure 5 outlines the process, which is described below.

Load Bugs. Firefox bug reports where extracted from the Bugzilla database. The data set includes bug metadata (title, creation time, etc.) and history (e.g., status change events with timestamps).

Extract Release Dates. Release dates were obtained from Mozilla's wiki [14]. Bugs reported until the release of version 4.0 were considered to be part of a traditional release; those reported after the release of 7.0 were considered to be part of a rapid release.

Parse Log. The central source code repository was downloaded and its commit log was parsed. For each commit, the time, the message, and the commit hash (i.e., the commit identifier) were extracted.

Map Commits to Bugs. Next, it was necessary to identify commits and map them to the bugs they fix. We read a sample of commit messages, and found them to be fairly consistent

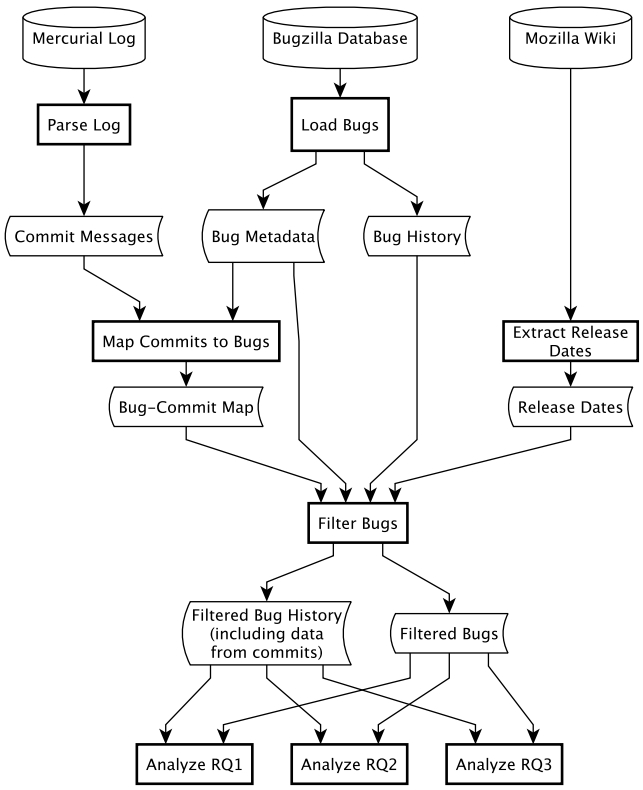

Fig. 5. Data analysis process

regarding the conventions adopted to indicate the bug being fixed. A typical commit message makes reference to the bug it fixes:

> Bug 939080 - Allow support-files in manifests to exist in parent paths; r=ted

For all commit messages starting with the word "bug" and a 5- or 6-digit number, the number was extracted and interpreted as the identifier of the bug fixed by the commit. Using this heuristic, about 74% of the commit messages could be traced back to bug reports[7]. A sample of such commit messages was read and no false positives were found.

We also needed to identify backout commits and determine which bug fixes were reverted by them. A typical backout commit references either the commit it reverts, the bug whose fix is being reverted, or both. A typical backout commit:

> Back out 7273dbeaeb88 (bug 157846) for mochitest and reftest bustage

All commits whose message contained the expression "back out" or a variation ("backout", "backed out", "backing out") were interpreted as backout commits. All decimal numbers with 5 to 6 digits were interpreted as bug report identifiers, and all hexadecimal numbers with 7 to 12 digits were interpreted as commit hashes.

Not all bug reports referenced in backout commits are bugs whose fix is being reverted, though. For instance, consider the following commit message:

> Back out parts of bug 698986 to resolve bug 716945

The commit message references two bug reports. The first one is the bug being reopened; the second one, though, is the the bug being fixed by the commit. Therefore, bug 716945 was not backed out (at least not in this particular commit).

After reading a sample of backout commits, we decided that bug identifiers after the following expressions do not refer to bugs being backed out:

- resolve, fix – e.g., "Backout bug 555133 to fix bug 555950." (i.e., bug 555133 was backed out, while bug 555950 was not);
- causing, cause, because – e.g., "Backed out changeset 705ef05105e8 for causing bug 503718 on OS X";
- due to – e.g., "Backed out changeset 58fd8a926bf5 (bug 366203) due to it causing bug 524293.";
- suspicion – e.g., "Backout revisions (...) on suspicion of causing (...) bug 536382."

Also, some backout commits do not reference bug identifiers; instead, they refer to another commit identifier. In this case, it is necessary to read the referenced commit to discover the bug it tried to fix. This bug is therefore considered reopened. For example:

> **commit b6d4...**: Bug 475968. Pad out the glyph extents of Windows text (...)
> **commit 980e...**: Backed out changeset b6d407af386f for causing (...)

In this example, one can deduce that bug 475968 was reopened, because commit *b6d4...*, that fixed it, was later backed out by commit `980e....`

Filter Bugs. Finally, the set of bugs was filtered. Only bugs that were referenced in commit messages were considered. The rationale is that these bugs involved a change in the source code.

Analyze RQ1. To investigate whether short releases impact the bug reopening rate, each bug was classified as reopened or not. A bug was considered reopened if the status of its bug report changed to REOPENED when the resolution was FIXED or if it was referenced in a backout commit. A contingency table was computed to count reopened and non-reopened bugs under traditional and rapid releases, and Fisher's exact test was applied to assess whether there is an association between reopening and release type.

Analyze RQ2. To investigate whether short releases impact the number of bugs reopened due to failing automated tests before a release, backout commits were scanned for the word "test". A reading of commit messages showed that the word "test" is associated with failing tests, crashing tests, or

[7]According to a previous study [6], about 33% of the issues classified as bugs are, in fact, feature requests, requests for performance improvements and other maintenance tasks. Therefore, it should *not* be inferred that corrective maintenance accounts for 74% of the commits.

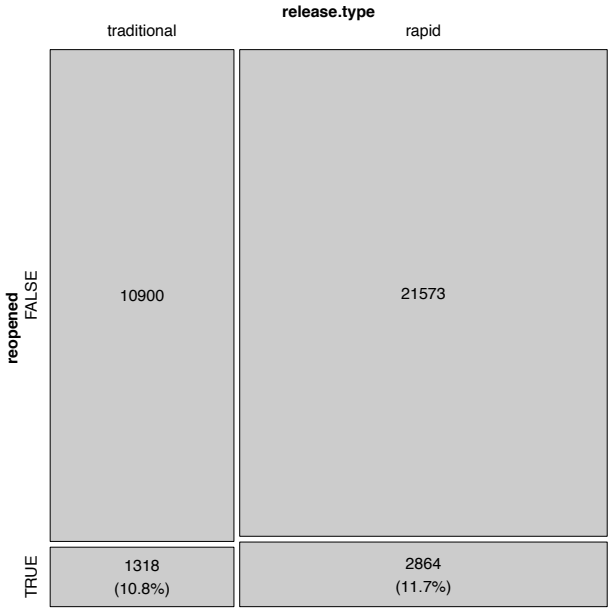

Fig. 6. Mosaic plot of bug reopening vs. release type

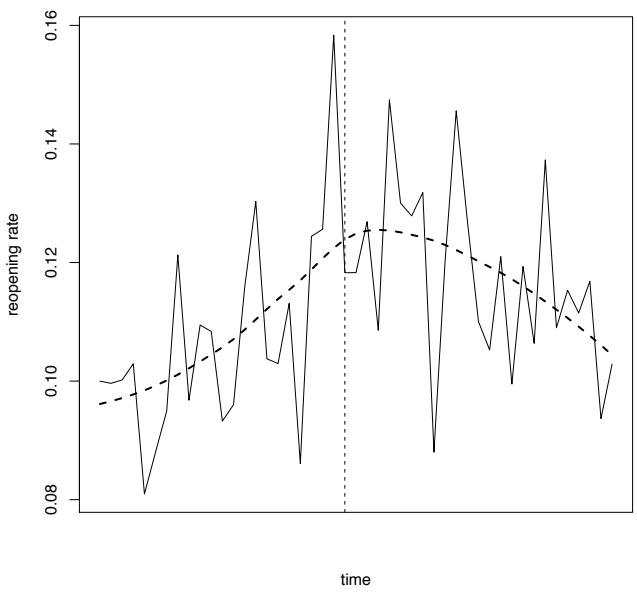

Fig. 7. Reopening rate over time

disturbance of testing infrastructure when it appears in backout commits. A contingency table was computed to count, within reopened bugs, those that were reopened due to failing tests under traditional and rapid releases, and Fisher's exact test was applied to assess whether there is an association between reopening due to tests and release type.

Analyze RQ3. To investigate whether short releases impact the time it takes for a bug to be reopened, also called latent time, all reopened bugs were considered. Then, the first reopening was selected (i.e., first status change to REOPENED or first backout commit, whichever comes first), as well as the last commit before the reopening. The time between those two events was computed as the latent time. Because latent times are not normally distributed, the latent times of traditional and short releases were compared using Mann-Whitney non-parametric test [17].

V. Results

This section shows the results found for each research question. A confidence level of 5% is assumed in all hypothesis tests.

RQ1: Do rapid releases impact the bug reopening rate?

After determining whether each bug reported in both traditional and rapid releases was reopened or not, the contingency table was computed. It can be visualized by the mosaic plot in Figure 6.

In the mosaic plot, each box represents a cell of the contingency table. The width of a box represents how many bugs there are for a certain release type. Boxes on the left side represent traditional releases, while boxes in the right side represent rapid releases. The height represents the proportion of bugs that were reopened (lower boxes) or not (upper boxes). The absolute numbers represent the number of bugs in each cell of the contingency table, and the percentages represent the reopening rate under each release type.

Although the two periods under analysis have similar length (approximately two years), it is easy to see in the mosaic plot that about twice as much bugs were fixed in the rapid release period. This finding may suggest that bugs are indeed fixed faster, which could contribute to increasing the bug reopening rate.

From Figure 6, it can be seen that the reopening rate increased from 12.6% to 13.5% after Mozilla adopted rapid releases, which represents a difference of 0.9 percentage points, or a relative change of approximately 7% ($100 \times 0.9/12.6$). This difference was found to be statistically significant at the 5% level ($p = 0.04965$).

It could be argued that the increased bug reopening rate was caused not by the switch to a different release type, but by metrics that tend to increase over time, such as the number of bugs reported and the software size. To help investigate this hypothesis, Figure 7 shows the reopening rate for each month in the period. To facilitate the analysis, the dashed line shows a local polynomial regression fitting (R function `loess`) of the original data, and the vertical line separates traditional and rapid releases. It can be seen that the reopening rate was not monotonically increasing in the period under analysis; therefore, there is no evidence that the observed difference in the bug reopening rate is related to software size or number of bugs. In fact, the correlation between reopening rate and number of bugs fixed is less than 0.1.

After moving to rapid releases, the bug reopening rate in Firefox increased 7%.

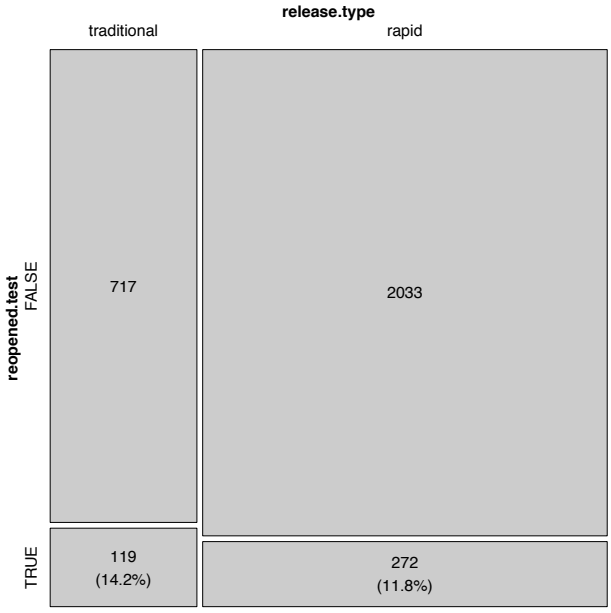

Fig. 8. Mosaic plot of bugs reopened due to failing automated tests (relative to total number of reopened bugs) vs. release type

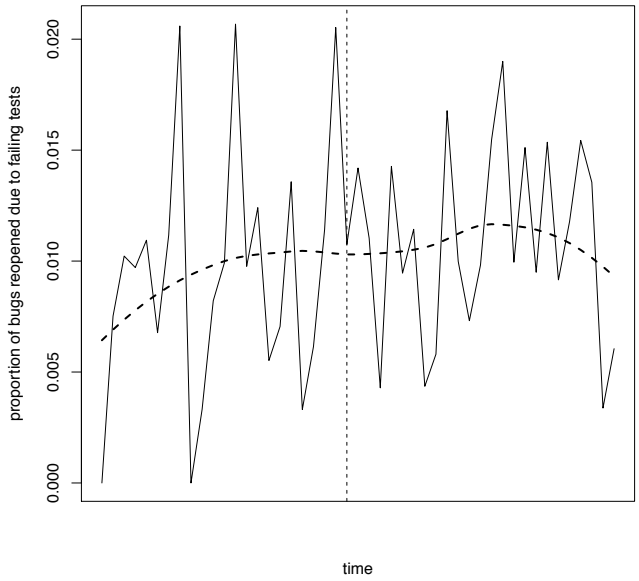

Fig. 9. Proportion of bugs reopened due to failing tests over time

RQ2: Do rapid releases impact the number of bugs reopened due to failing automated tests?

Figure 8 shows a mosaic plot of the number of bugs reopened because of failing tests (relative to the number of reopened bugs) under both release types. Although the proportion of these bugs is smaller under rapid releases, the difference is not statistically significant ($p = 0.67$).

Figure 9 shows the same data plotted for each month in the period under analysis. No clear trend can be inferred from the time series.

There is no significant difference in the proportion of bugs reopened due to failing tests under rapid releases.

RQ3: Do rapid releases impact the time it takes for a bug to be reopened?

Results for RQ1 showed that, under rapid releases, bugs are fixed faster. Figure 10 shows the time-to-fix for each bug, i.e., the number of days from the creation of a bug report until the first fix is submitted. We consider a fix when a bug is marked FIXED or when a commit references the bug, whichever comes first. The median time-to-fix decreased from 17 to 7 days. The difference is statistically significant ($p < 0.001$).

Latent time, i.e., the time it takes for a bug report to change its status from RESOLVED-FIXED to REOPENED, is shown in Figure 11, in logarithmic scale. The median latent time is low: 4.8 hours for traditional releases and 3.5 hours for rapid releases. The difference is not statistically significant ($p > 0.05$), therefore one cannot say that the typical latent time is smaller under rapid releases.

If one is interested in extreme cases, however, the difference is more accentuated. Figure 12 shows the top 10% latent

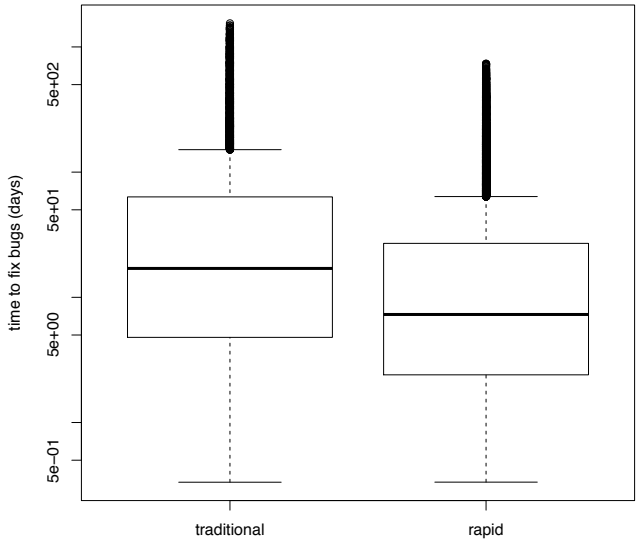

Fig. 10. Time-to-fix under traditional and rapid releases

times for both release types, measured in days. In this subset, the median decreased from 48 to 27 days after moving to rapid releases ($p < 0.001$).

This difference can probably be explained by the fact that Firefox is released more often. Because bugs that are reopened within a 6-week cycle do not reach end users, it can be expected that the latent time for those bugs is not impacted by rapid releases.

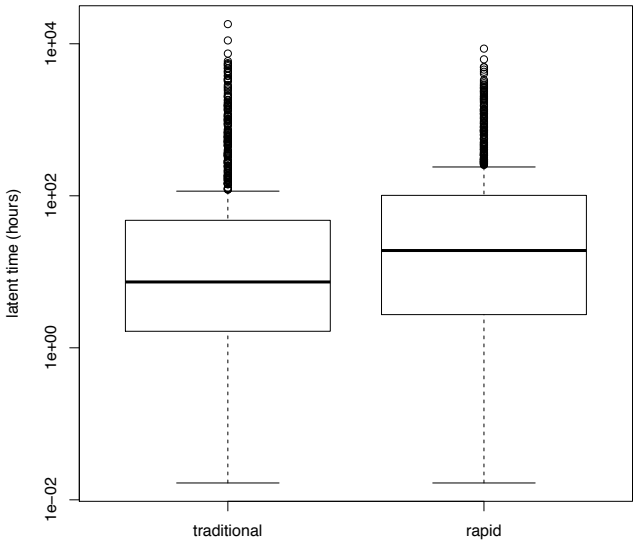

Fig. 11. Latent time (time-to-reopen) under traditional and rapid releases

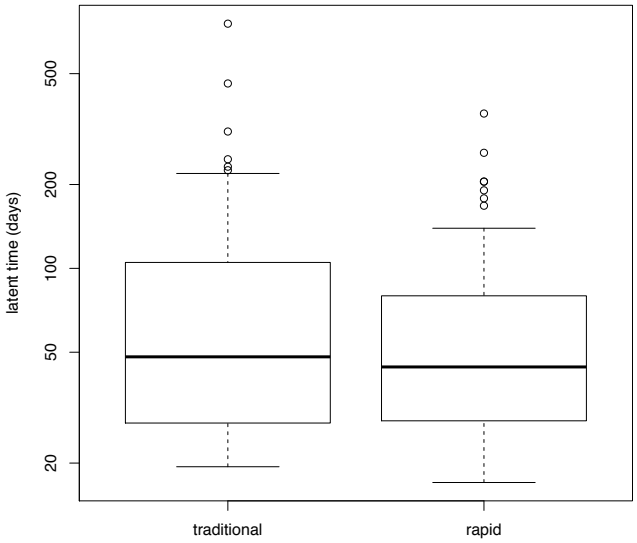

Fig. 12. Topmost 10% latent times under traditional and rapid releases

While typical bugs are reopened within hours under both traditional and rapid releases, long latent times (measured in weeks and months) tend to be shorter under rapid releases.

VI. Lessons Learned and Threats to Validity

Bug and source code repositories from open source projects are a great opportunity for software engineering researchers to study both the development process and the artifacts of real-world, global-scale software. At the same time, because processes are complex, varied, and may evolve, it is a challenge to extract useful and reliable information from software repositories.

As showed in the previous section, first attempts to address the research questions had serious limitations, which ultimately led to contradictory answers. Hopefully, most limitations were discovered and overcome, thanks to three practices that are commented below:

- **Study the development process**. It involves reading wiki pages aimed at developers and other contributors, but also slide decks and and tool documentation (e.g., Bugzilla, Mercurial).
- **Explore the data**. By computing summary statistics and plotting time series, much information can be gathered about process changes and data noise. Also, it is important to read some bug reports and commit logs to get familiar with conventions and to test assumptions.
- **Get feedback on preliminary results**. It is important to show preliminary research results to the developers. Because they work daily with the data being analyzed, they can point to limitations in the study design and explain details in the process that are not documented anywhere else.

Getting feedback from the community is not trivial, since developers usually do not have any incentive to cooperate with researchers. In this research, we were able to reach them and obtain valuable feedback, and we attribute this success to three factors:

- **relevance**: this research is about something that matters to them; rapid releases and bug reopening impact their daily work;
- **language**: questions and results were explained using their vocabulary, and not based on theoretical concepts and terms.
- **level of detail**: results were presented clearly and succintly, but some level of detail was helpful; some contributors, specially quality engineers, are interested in knowing the data sources you used and how metrics were computed; without such data, they cannot indicate limitations in the data analysis.

Even though this study benefited from data exploration and community feedback, it is still subject to threats that cannot be entirely overcome without a great amount of effort. The threats are listed below, together with measures taken to mitigate them.

Construct validity (to which extent the study measures what it intends to measure). The detection of bug reopening was based in heuristics, including pattern matching on commit messages. Although numerous messages were read in order to determine recurrent patterns to help extract identifiers of reopened bugs, not all commits conform to the most common patterns. It is possible that the actual bug reopening rate is different from what was measured, although we believe that the deviance would not be large. Also, what matters in this study is the *difference* between the bug reopening rate under

two release types, a metric that is less influenced by systematic bias.

Also, to detect bugs reopened due to failing tests, commit messages were scanned for the word "test", a heuristic that relies on developers following implicit conventions. When asked about the heuristic, a Mozilla engineer said that, although not all test failures are documented in commit messages, the word "test", in this context, most likely refers to failing automated tests. Therefore, cases detected by the heuristic can be considered a sample of all reopenings caused by failing tests (although the available data is not enough to determine whether the sample is biased).

Internal validity (to which extent conclusions can be made from what was measured). The observed differences under traditional and rapid releases could be attributed to factors other than release type, since the development process may have changed over time in many different ways. For example, a Mozilla engineer pointed out that the criteria to determine when a bug should be reopened and when a new bug report should be filled can change over time and influence the analysis of bug reopening.

External validity (to which extent results can be generalized). All the conclusions in this study were based on data from one single open source project. The results are not expected to be generalizable to other projects; instead, the purpose of this study is to provide insights on the questions beings studied. Furthermore, it is not trivial to extend the study to other projects, since the projects must have migrated from a traditional to a rapid release model. Also, the study requires a high level of understanding about the process and how it is recorded by tools.

VII. Conclusion

Although the benefits of rapid releases are clear, there is little empirical knowledge about their impacts on the development process. In this study, we showed how rapid releases affect bug reopening, which causes rework and thus reduces productivity.

At least in Firefox, the move to rapid releases was associated with an increase in the bug reopening rate. The difference, though, was of just 7%. The tighter deadlines for testing, which would supposedly increase the reopening rate, could have been offset by improvements in the development infrastructure and by the increase of specialized testers [7].

On the other hand, faulty bug fixes that used to be latent for long periods tended to be discovered earlier under rapid releases, possibly because of feedback from early adopters. Firefox policy to make alpha (aurora) and beta versions available before releasing the final version contributes to reduce the number of faulty bug fixes that are shipped to end users.

Acknowledgement

The authors would like to thank all Mozilla engineers who provided feedback on preliminary results of this research. Special thanks to Lucas Rocha, for patiently explaining Mozilla's release process, and to Mike Hoye, for making Mozilla's Bugzilla database dump available.

References

[1] R. Baskerville and J. Pries-Heje, "Short cycle time systems development." *Inf. Syst. J.*, vol. 14, no. 3, pp. 237–264, 2004. [Online]. Available: http://dblp.uni-trier.de/db/journals/isj/isj14.html#BaskervilleP04

[2] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox." in *MSR*, M. Lanza, M. D. Pent, and T. Xi, Eds. IEEE, 2012, pp. 179–188. [Online]. Available: http://dblp.uni-trier.de/db/conf/msr/msr2012.html#KhomhDZA12

[3] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 1074–1083. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337363

[4] R. Patton, *Software Testing (2nd Edition)*. Indianapolis, IN, USA: Sams, 2005.

[5] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, ser. CASCON '08. New York, NY, USA: ACM, 2008, pp. 23:304–23:318. [Online]. Available: http://doi.acm.org/10.1145/1463788.1463819

[6] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," Universitt des Saarlandes, Saarbrcken, Germany, Tech. Rep., August 2012.

[7] M. Mantyla, F. Khomh, B. Adams, E. Engstrom, and K. Petersen, "On rapid releases and software testing," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, Sept 2013, pp. 20–29.

[8] Jono, "Everybody hates firefox updates," 2012, http://evilbrainjono.net/blog?permalink=1094.

[9] S. Dean, "Firefox's move to a rapid release cycle didn't just upset users," 2013, http://ostatic.com/blog/firefoxs-move-to-a-rapid-release-cycle-didnt-just-upset-users.

[10] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *Proceedings of the 2010 17th Working Conference on Reverse Engineering*, ser. WCRE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 249–258. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2010.36

[11] E. Shihab, "An exploration of challenges limiting pragmatic software defect prediction," Ph.D. dissertation, School of Computing, Queen's University, Kingston, Ontario, Canada, 2012.

[12] A. Almossawi, "Investigating the architectural drivers of defects in open-source software systems: an empirical study of defects and reopened defects in gnome," 2012.

[13] A. Jongyindee, M. Ohira, A. Ihara, and K.-i. Matsumoto, "Good or bad committers? a case study of committers' cautiousness and the consequences on the bug fixing process in the Eclipse project," in *Proc. of the 2011 Joint Conf. of the 21st International Workshop on Softw. Measurement and the 6th International Conf. on Softw. Process and Product Measurement*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 116–125. [Online]. Available: http://dx.doi.org/10.1109/IWSM-MENSURA.2011.24

[14] Mozilla, "Mozillawiki: Releases," 2014, https://wiki.mozilla.org/Releases.

[15] ——, "Mozilla release processes," 2011, http://mozilla.github.io/process-releases/.

[16] R. Souza, C. Chavez, and R. Bittencourt, "Patterns for cleaning up bug data," in *Proc. of the 1st Workshop on Data Analysis Patterns in Softw. Engineering*. IEEE, May 2013.

[17] J. McDonald and U. of Delaware, *Handbook of Biological Statistics*. Sparky House Publishing, 2009. [Online]. Available: http://books.google.com.br/books?id=AsRTywAACAAJ